Architektury systemów komputerowych

Lista zadań nr 5

Na zajęcia 29 marca i 4 kwietnia 2023

Uwaga! Należy być przygotowanym do wyjaśnienia semantyki każdej instrukcji, która pojawia się w treści zadania. W tym celu posłuż się dokumentacją: x86 and amd64 instruction reference[1]. W szczególności trzeba wiedzieć jak dana instrukcja korzysta z rejestru flag «EFLAGS» tam, gdzie obliczenia zależą od jego wartości.

W trakcie tłumaczeniu kodu z asemblera x86-64 do języka C należy trzymać się następujących wytycznych:

- Używaj złożonych wyrażeń minimalizując liczbę zmiennych tymczasowych.
- Nazwy wprowadzonych zmiennych muszą opisywać ich zastosowanie, np. `result` zamiast `rax`.
- Instrukcja `goto` jest zabroniona. Należy używać instrukcji sterowania `if`, `for`, `while` i `switch`.
- Pętle «`while`» należy przetłumaczyć do pętli «`for`», jeśli poprawia to czytelność kodu.

Zadanie 1. Zaimplementuj poniższą funkcję w asemblerze x86-64. Wartości «x» i «y» typu «`uint64_t`» są przekazywane przez rejestry `%rdi` i `%rsi`, a wynik zwracany w rejestrze `%rax`. Najpierw rozwiąż zadanie używając instrukcji skoku warunkowego. Potem przepisz je używając instrukcji «`sbb`».

$$addu(x,y) = \begin{cases} \texttt{ULONG\_MAX} & \text{dla } x+y \geq \texttt{ULONG\_MAX} \\ x+y & \text{w p.p.} \end{cases}$$

Wskazówka! Rozwiązanie wzorcowe składa się z 3 instrukcji bez «`ret`».

Zadanie 2. Zaimplementuj funkcję zdefiniowaną poniżej w asemblerze x86-64. Taka procedura w języku C miałaby sygnaturę «`long cmp(uint64_t x, uint64_t y)`».

$$cmp(x,y) = \begin{cases} -1 & \text{gdy } x < y \\ 1 & \text{gdy } x > y \\ 0 & \text{gdy } x = y \end{cases}$$

Wskazówka: Rozwiązanie wzorcowe ma cztery wiersze (bez `ret`). Użyj instrukcji `adc`, `sbb` i `neg`.

Zadanie 3. Zapisz w języku C funkcję o sygnaturze «`int puzzle(long x, unsigned n)`» której kod w asemblerze podano niżej. Zakładamy, że parametr «`n`» jest niewiększy niż 64. Przedstaw jednym zdaniem co robi ta procedura.

```
1  puzzle: testl %esi, %esi
2          je    .L4
3          xorl  %edx, %edx
4          xorl  %eax, %eax
5  .L3:    movl  %edi, %ecx
6          andl  $1, %ecx
7          addl  %ecx, %eax
8          sarq  %rdi
9          incl  %edx
10         cmpl  %edx, %esi
11         jne   .L3
12         ret
13 .L4:    movl  %esi, %eax
14         ret
```

Uwaga! Instrukcja zapisująca młodszą połowę 64-bitowego rejestru ustawia na 0 jego starszą połowę (brzydota x86-64).

Zadanie 4. Poniżej zamieszczono kod procedury o sygnaturze «`long puzzle2(char *s, char *d)`». Wyznacz *bloki podstawowe* oraz narysuj *graf przepływu sterowania*. Przetłumacz tę procedurę na język C, a następnie jednym zdaniem powiedz co ona robi.

[1] `http://www.felixcloutier.com/x86/`

```
puzzle2:
        movq  %rdi, %rax
.L3:    movb  (%rax), %r9b
        leaq  1(%rax), %r8
        movq  %rsi, %rdx
.L2:    movb  (%rdx), %cl
        incq  %rdx
        testb %cl, %cl
        je    .L4
        cmpb  %cl, %r9b
        jne   .L2
        movq  %r8, %rax
        jmp   .L3
.L4:    subq  %rdi, %rax
        ret
```

Zadanie 5 (2). Poniżej widnieje kod funkcji o sygnaturze «`uint32_t puzzle3(uint32_t n, uint32_t d)`». Wyznacz bloki podstawowe oraz narysuj graf przepływu sterowania, po czym przetłumacz tę funkcję na język `C`. Na podstawie ustępu *„Mixing C and Assembly Language”* strony GNU Assembler Examples[2] napisz i zaprezentuj działanie programu, który pomógł Ci odpowiedzieć na pytanie co ta funkcja robi.

```
puzzle3:
        movl  %edi, %edi
        salq  $32, %rsi
        movl  $32, %edx
        movl  $0x80000000, %ecx
        xorl  %eax, %eax
.L3:    addq  %rdi, %rdi
        movq  %rdi, %r8
        subq  %rsi, %r8
        js    .L2
        orl   %ecx, %eax
        movq  %r8, %rdi
.L2:    shrl  %ecx
        decl  %edx
        jne   .L3
        ret
```

Zadanie 6 (2). Poniżej zamieszczono kod rekurencyjnej procedury o sygnaturze «`long puzzle4(long *a, long v, uint64_t s, uint64_t e)`». Wyznacz bloki podstawowe oraz narysuj graf przepływu sterowania. Przetłumacz tę procedurę na język `C`, a następnie jednym zdaniem powiedz co ona robi.

```
puzzle4:
        movq  %rcx, %rax
        subq  %rdx, %rax
        shrq  %rax
        addq  %rdx, %rax
        cmpq  %rdx, %rcx
        jb    .L5
        movq  (%rdi,%rax,8), %r8
        cmpq  %rsi, %r8
        je    .L10
        jg    .L11
        leaq  1(%rax), %rdx
        call  puzzle4
.L10:   ret
.L11:   leaq  -1(%rax), %rcx
        call  puzzle4
        ret
.L5:    movq  $-1, %eax
        ret
```

Wskazówka: Z reguły procedurę «`puzzle4`» woła się następująco: «`i = puzzle4(a, v, 0, n - 1)`».

Zadanie 7 (2). Poniższy kod w asemblerze otrzymano w wyniku deasemblacji funkcji zadeklarowanej jako «`long switch_prob(long x, long n)`». Zapisz w języku `C` kod odpowiadający tej funkcji.

```
400590 <switch_prob>:
400590: 48 83 ef 3c             subq  $0x3c,%rsi
400594: 48 83 fe 05             cmpq  $0x5,%rsi
400598: 77 29                   ja    *0x4005c3
40059a: ff 24 f5 f8 06 40 00    jmpq  *0x4006f8(,%rsi,8)
4005a1: 48 8d 04 fd 00 00 00 00 lea   0x0(,%rdi,8),%rax
4005a9: c3                      retq
4005aa: 48 89 f8                movq  %rdi,%rax
4005ad: 48 c1 f8 03             sarq  $0x3,%rax
4005b1: c3                      retq
4005b2: 48 89 f8                movq  %rdi,%rax
4005b5: 48 c1 e0 04             shlq  $0x4,%rax
4005b9: 48 29 f8                subq  %rdi,%rax
4005bc: 48 89 c7                movq  %rax,%rdi
4005bf: 48 0f af ff             imulq %rdi,%rdi
4005c3: 48 8d 47 4b             leaq  0x4b(%rdi),%rax
4005c7: c3                      retq
```

Zrzut pamięci przechowującej tablicę skoków:

```
(gdb) x/6gx 0x4006f8
0x4006f8: 0x4005a1
0x400700: 0x4005a1
0x400708: 0x4005b2
0x400710: 0x4005c3
0x400718: 0x4005aa
0x400720: 0x4005bf
```

[2] http://cs.lmu.edu/~ray/notes/gasexamples/